Recommendations_with_IBM

June 5, 2022

1 Recommendations with IBM

In this notebook, you will be putting your recommendation skills to use on real data from the IBM Watson Studio platform.

You may either submit your notebook through the workspace here, or you may work from your local machine and submit through the next page. Either way assure that your code passes the project RUBRIC. **Please save regularly.**

By following the table of contents, you will build out a number of different methods for making recommendations that can be used for different situations.

1.1 Table of Contents

I. Section **??** II. Section **??** III. Section **??** IV. Section **??** V. Section **??** VI. Section **??**

At the end of the notebook, you will find directions for how to submit your work. Let's get started by importing the necessary libraries and reading in the data.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t
import pickle

%matplotlib inline

df = pd.read_csv('data/user-item-interactions.csv')
df_content = pd.read_csv('data/articles_community.csv')
del df['Unnamed: 0']
del df_content['Unnamed: 0']
```

[2]:
```
# Show df to get an idea of the data
df.tail()
```

[2]:
```
       article_id                                              title  \
45988      1324.0                      ibm watson facebook posts for 2015
45989       142.0  neural networks for beginners: popular types a…
```

```
45990       233.0     bayesian nonparametric models - stats and bots
45991      1160.0       analyze accident reports on amazon emr spark
45992        16.0   higher-order logistic regression for large dat…

                                          email
45988  d21b998d7a4722310ceeaa3c6aaa181a36db2d73
45989  d21b998d7a4722310ceeaa3c6aaa181a36db2d73
45990  4faeed980a7cd11e0f3cf2058cc04daa2ef11452
45991  abbf639ba05daa5249c520e290283a6d726ba78d
45992  1f18e8aaccd6c8720180c3fe264c8aef5b00697f
```

[3]:

```python
# Show df_content to get an idea of the data
df_content.head()
```

[3]:

```
                                            doc_body  \
0  Skip navigation Sign in SearchLoading…\r\n\r…
1  No Free Hunch Navigation * kaggle.com\r\n\r\n …
2    * Login\r\n * Sign Up\r\n\r\n * Learning Pat…
3  DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA…
4  Skip navigation Sign in SearchLoading…\r\n\r…

                                     doc_description  \
0  Detect bad readings in real time using Python …
1  See the forest, see the trees. Here lies the c…
2  Here’s this week’s news in Data Science and Bi…
3  Learn how distributed DBs solve the problem of…
4  This video demonstrates the power of IBM DataS…

                                       doc_full_name doc_status  article_id
0  Detect Malfunctioning IoT Sensors with Streami…       Live           0
1  Communicating data science: A guide to present…       Live           1
2         This Week in Data Science (April 18, 2017)       Live           2
3  DataLayer Conference: Boost the performance of…       Live           3
4      Analyze NY Restaurant data using Spark in DSX       Live           4
```

1.1.1 At first glance article_id column is the only column that seems common between the two dataframes. Further more the article_id in df is a float where as it is a integer in df_content.

1.1.2 I’m not sure but i think the doc_full_name column might be similar to the title column in df.

[4]:

```python
df_content[df_content['article_id']==16]
```

[4]:

```
                                             doc_body  \
16  * Home\r\n * Research\r\n * Partnerships and C…
```

```
                                      doc_description  \
16  The performance of supervised predictive model…

                                        doc_full_name doc_status  article_id
16  Higher-order Logistic Regression for Large Dat…       Live          16
```

1.1.3 Yup it seems like doc_full_name and title are the same column with minor differences, like, the title in df is all lower case

1.1.4 Also another point worth noting is that there are some article id's that are present in df but not in df_content. The df_content have articles id's numbered from 0 to 1050 where as df contains article_id's which are way beyond 1050. The df_content dataframe seems like a collection of articles present in the community and it seemed like users can interact with only those articles but from this observation this does not seem correct. So for now i do not know how to deal with that.

1.1.5 Part I : Exploratory Data Analysis

Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

`1.` What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

[5]:
```
user_item_count = df.groupby('email')['article_id'].count().reset_index()
```

[6]:
```
user_item_count.max()
```

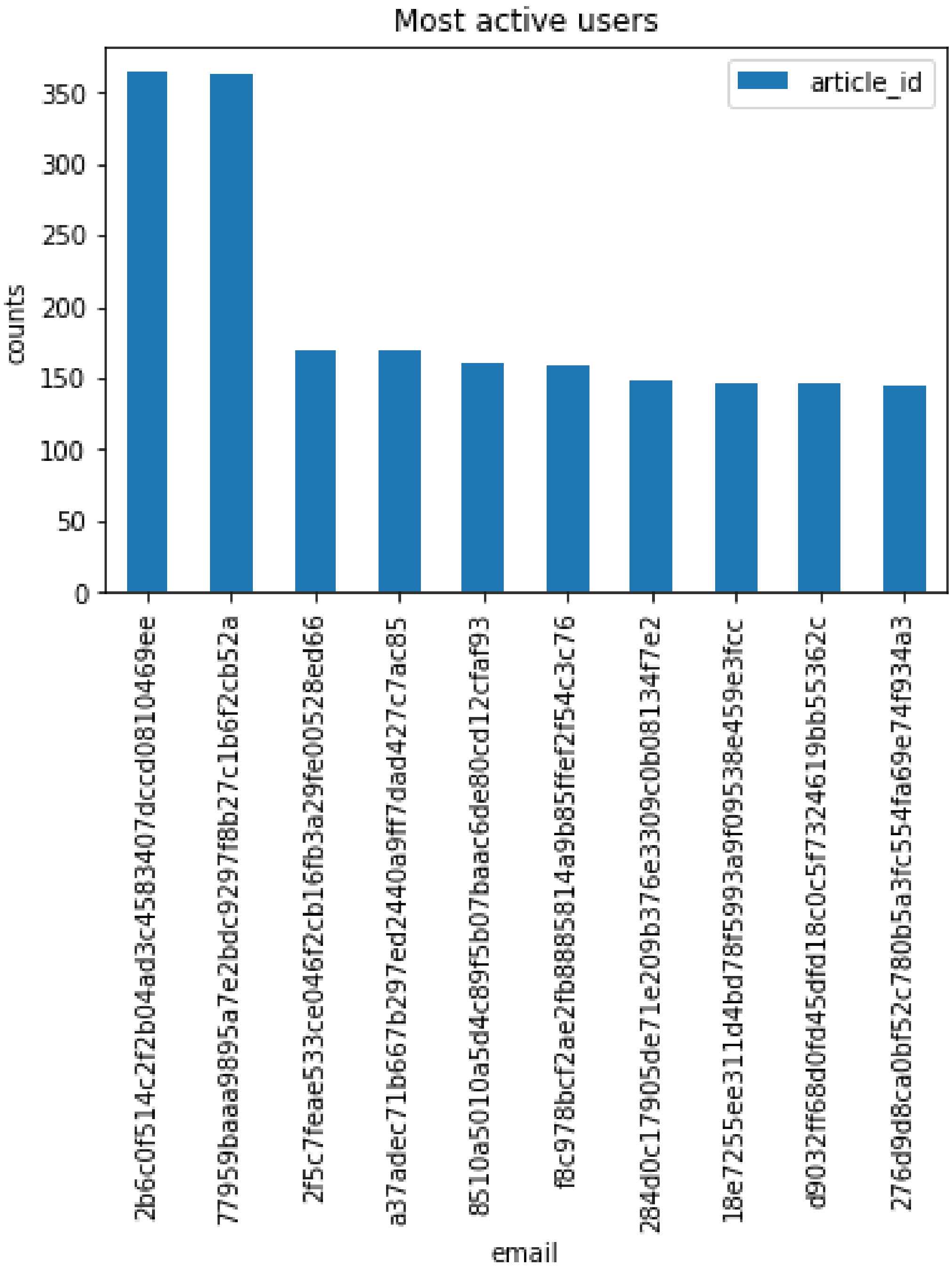
[6]:
```
email         fffb93a166547448a0ff0232558118d59395fecd
article_id                                         364
dtype: object
```

[7]:
```
user_item_count.median()
```

[7]:
```
article_id    3.0
dtype: float64
```

[8]:
```
# Fill in the median and maximum number of user_article interactios below

median_val = 3# 50% of individuals interact with ____ number of articles or␣
 ↪fewer.
max_views_by_user = 364# The maximum number of user-article interactions by any␣
 ↪1 user is ______.
```

[9]:
```
user_item_count.sort_values(by='article_id',ascending=False)[:10].plot.
 ↪bar(x='email',y='article_id');
plt.ylabel('counts');
plt.title('Most active users');
```



2. Explore and remove duplicate articles from the **df_content** dataframe.

[10]:
```python
# Find and explore duplicate articles
df_content.shape[0]
```

[10]: 1056

[11]:
```python
df_content['article_id'].nunique()
```

[11]: 1051

[12]:
```python
# Remove any rows that have the same article_id - only keep the first
df_content['article_id'].value_counts()[:10]
```

[12]:
```
221    2
232    2
577    2
398    2
50     2
356    1
355    1
354    1
353    1
345    1
Name: article_id, dtype: int64
```

[13]:
```python
df_content.drop_duplicates(subset = 'article_id', inplace=True)
```

[14]:
```python
df_content.shape[0]
```

[14]: 1051

3. Use the cells below to find:

a. The number of unique articles that have an interaction with a user.
b. The number of unique articles in the dataset (whether they have any interactions or not). **c.** The number of unique users in the dataset. (excluding null values) **d.** The number of user-article interactions in the dataset.

[15]:
```python
df.shape[0]
```

[15]: 45993

[16]:
```python
df['article_id'].nunique()
```

[16]: 714

[17]:
```python
df['email'].nunique()
```

[17]: 5148

```
[18]: unique_articles = 714# The number of unique articles that have at least one␣
      ↪interaction
      total_articles = 1051# The number of unique articles on the IBM platform
      unique_users = 5148# The number of unique users
      user_article_interactions = 45993# The number of user-article interactions
```

4. Use the cells below to find the most viewed **article_id**, as well as how often it was viewed. After talking to the company leaders, the `email_mapper` function was deemed a reasonable way to map users to ids. There were a small number of null values, and it was found that all of these null values likely belonged to a single user (which is how they are stored using the function below).

```
[19]: article_frequency = df.groupby('article_id')['email'].count().
      ↪sort_values(ascending=False).reset_index()
```

```
[20]: article_frequency.columns=['article_id', 'freq']
```

```
[21]: article_frequency.loc[0,'freq']
```

```
[21]: 937
```

```
[22]: article_frequency.loc[0,'article_id']
```

```
[22]: 1429.0
```

```
[23]: most_viewed_article_id = '1429.0'# The most viewed article in the dataset as a␣
      ↪string with one value following the decimal
      max_views = 937# The most viewed article in the dataset was viewed how many␣
      ↪times?
```

```
[24]: ## No need to change the code here - this will be helpful for later parts of␣
      ↪the notebook
      # Run this cell to map the user email to a user_id column and remove the email␣
      ↪column

      def email_mapper():
          coded_dict = dict()
          cter = 1
          email_encoded = []

          for val in df['email']:
              if val not in coded_dict:
                  coded_dict[val] = cter
                  cter+=1

              email_encoded.append(coded_dict[val])
          return email_encoded
```

```
email_encoded = email_mapper()
del df['email']
df['user_id'] = email_encoded

# show header
df.head()
```

[24]:

	article_id	title	user_id
0	1430.0	using pixiedust for fast, flexible, and easier…	1
1	1314.0	healthcare python streaming application demo	2
2	1429.0	use deep learning for image classification	3
3	1338.0	ml optimization using cognitive assistant	4
4	1276.0	deploy your python model as a restful api	5

[25]:

```
## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell

sol_1_dict = {
    '`50% of individuals have _____ or fewer interactions.`': median_val,
    '`The total number of user-article interactions in the dataset is ______.`':
 ↪ user_article_interactions,
    '`The maximum number of user-article interactions by any 1 user is ______.
 ↪`': max_views_by_user,
    '`The most viewed article in the dataset was viewed _____ times.`':␣
 ↪max_views,
    '`The article_id of the most viewed article is ______.`':␣
 ↪most_viewed_article_id,
    '`The number of unique articles that have at least 1 rating ______.`':␣
 ↪unique_articles,
    '`The number of unique users in the dataset is ______`': unique_users,
    '`The number of unique articles on the IBM platform`': total_articles
}

# Test your dictionary against the solution
t.sol_1_test(sol_1_dict)
```

```
It looks like you have everything right here! Nice job!
```

1.1.6 Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

1. Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

[26]:
```
df.groupby('article_id')['title'].count().sort_values(ascending=False).
↪reset_index()['article_id'][0:10]
```

[26]:
```
0    1429.0
1    1330.0
2    1431.0
3    1427.0
4    1364.0
5    1314.0
6    1293.0
7    1170.0
8    1162.0
9    1304.0
Name: article_id, dtype: float64
```

[27]:
```
def get_top_articles(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    top_articles = list(df.groupby('title')['article_id'].count().
↪sort_values(ascending=False).reset_index()['title'][0:n])


    return top_articles # Return the top article titles from df (not df_content)

def get_top_article_ids(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    top_articles = list(df.groupby('article_id')['title'].count().
↪sort_values(ascending=False).reset_index()['article_id'][0:n])
    top_articles = list(map(str,top_articles))

    return top_articles # Return the top article ids
```

[28]:
```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car
accident reports', 'visualize car data with brunel', 'use xgboost, scikit-learn
& ibm watson machine learning apis', 'predicting churn with the spss random tree
algorithm', 'healthcare python streaming application demo', 'finding optimal
locations of new store using decision optimization', 'apache spark lab, part 1:
basic concepts', 'analyze energy consumption in buildings', 'gosales
transactions for logistic regression model']
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0',
'1162.0', '1304.0']
```

[29]:
```
# Test your function by returning the top 5, 10, and 20 articles
top_5 = get_top_articles(5)
top_10 = get_top_articles(10)
top_20 = get_top_articles(20)

# Test each of your three lists from above
t.sol_2_test(get_top_articles)
```

```
Your top_5 looks like the solution list! Nice job.
Your top_10 looks like the solution list! Nice job.
Your top_20 looks like the solution list! Nice job.
```

1.1.7 Part III: User-User Based Collaborative Filtering

`1.` Use the function below to reformat the **df** dataframe to be shaped with users as the rows and articles as the columns.

- Each **user** should only appear in each **row** once.
- Each **article** should only show up in one **column**.
- **If a user has interacted with an article, then place a 1 where the user-row meets for that article-column**. It does not matter how many times a user has interacted with the article, all entries where a user has interacted with an article should be a 1.
- **If a user has not interacted with an item, then place a zero where the user-row meets for that article-column**.

Use the tests to make sure the basic structure of your matrix matches what is expected by the solution.

[30]:
```
# create the user-article matrix with 1's and 0's

def create_user_item_matrix(df):
    '''
    INPUT:
    df - pandas dataframe with article_id, title, user_id columns
```

```
    OUTPUT:
    user_item - user item matrix

    Description:
    Return a matrix with user ids as rows and article ids on the columns with 1␣
↪values where a user interacted with
    an article and a 0 otherwise
    '''
    user_item=df.drop_duplicates(subset=['article_id', 'user_id']).
↪groupby(['user_id', 'article_id'])['title'].count().unstack(fill_value=0)

    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

[31]:
```
user_item
```

[31]:
```
article_id  0.0     2.0     4.0     8.0     9.0     12.0    14.0    15.0    \
user_id
1                0       0       0       0       0       0       0       0
2                0       0       0       0       0       0       0       0
3                0       0       0       0       0       1       0       0
4                0       0       0       0       0       0       0       0
5                0       0       0       0       0       0       0       0
…              …     …     …     …     …     …     …     …
5145             0       0       0       0       0       0       0       0
5146             0       0       0       0       0       0       0       0
5147             0       0       0       0       0       0       0       0
5148             0       0       0       0       0       0       0       0
5149             0       0       0       0       0       0       0       0

article_id  16.0    18.0    …  1434.0  1435.0  1436.0  1437.0  1439.0  \
user_id                     …
1                0       0  …       0       0       1       0       1
2                0       0  …       0       0       0       0       0
3                0       0  …       0       0       1       0       0
4                0       0  …       0       0       0       0       0
5                0       0  …       0       0       0       0       0
…              …     …  …  …     …     …     …     …
5145             0       0  …       0       0       0       0       0
5146             0       0  …       0       0       0       0       0
5147             0       0  …       0       0       0       0       0
5148             0       0  …       0       0       0       0       0
5149             1       0  …       0       0       0       0       0

article_id  1440.0  1441.0  1442.0  1443.0  1444.0
```

```
user_id
1                 0       0       0       0       0
2                 0       0       0       0       0
3                 0       0       0       0       0
4                 0       0       0       0       0
5                 0       0       0       0       0
…             …      …      …      …      …
5145              0       0       0       0       0
5146              0       0       0       0       0
5147              0       0       0       0       0
5148              0       0       0       0       0
5149              0       0       0       0       0

[5149 rows x 714 columns]
```

[32]:

```python
## Tests: You should just need to run this cell.  Don't change the code.
assert user_item.shape[0] == 5149, "Oops!  The number of users in the␣
↪user-article matrix doesn't look right."
assert user_item.shape[1] == 714, "Oops!  The number of articles in the␣
↪user-article matrix doesn't look right."
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by␣
↪user 1 doesn't look right."
print("You have passed our quick tests!  Please proceed!")
```

```
You have passed our quick tests!  Please proceed!
```

2. Complete the function below which should take a user_id and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided user_id, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

[33]:

```python
# compute similarity of each user to the provided user
dot_prod_user= user_item.dot(np.transpose(user_item))


def find_similar_users(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
    similar_users - (list) an ordered list where the closest users (largest dot␣
↪product users)
```

```
                     are listed first

    Description:
    Computes the similarity of every pair of users based on the dot product
    Returns an ordered

    '''

    # sort by similarity
    # create list of just the ids
    # this one line does all of the above
    most_similar_users= list(np.argsort(dot_prod_user[user_id])[::-1]+1)
    # remove the own user's id
    most_similar_users.remove(user_id)




    return most_similar_users # return a list of the users in order from most␣
↪to least similar
```

[34]:
```
# Do a spot check of your function
print("The 10 most similar users to user 1 are: {}".
 ↪format(find_similar_users(1)[:10]))
print("The 5 most similar users to user 3933 are: {}".
 ↪format(find_similar_users(3933)[:5]))
print("The 3 most similar users to user 46 are: {}".
 ↪format(find_similar_users(46)[:3]))
```

```
The 10 most similar users to user 1 are: [3933, 23, 3782, 203, 4459, 131, 3870,
46, 4201, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 4459, 203]
The 3 most similar users to user 46 are: [4201, 23, 3782]
```

3. Now that you have a function that provides the most similar users to each user, you will want to use these users to find articles you can recommend. Complete the functions below to return the articles you would recommend to each user.

[35]:
```
def get_article_names(article_ids, df=df):
    '''
    INPUT:
    article_ids - (list) a list of article ids
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
```

```
    article_names - (list) a list of article names associated with the list of␣
↪article ids
                    (this is identified by the title column)
    '''
    # Your code here
    article_names = df[df['article_id'].isin(article_ids)]['title'].unique().
↪tolist()


    return article_names # Return the article names associated with list of␣
↪article ids


def get_user_articles(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
    article_ids - (list) a list of the article ids seen by the user
    article_names - (list) a list of article names associated with the list of␣
↪article ids
                    (this is identified by the doc_full_name column in␣
↪df_content)

    Description:
    Provides a list of the article_ids and article titles that have been seen␣
↪by a user
    '''
    # Your code here
    temp=(user_item.loc[user_id]==1).to_frame().reset_index()
    temp.columns=['article_id','val']
    article_ids=temp[temp['val']==True]['article_id'].tolist()
    article_ids = list(map(str,article_ids))
    article_names = get_article_names(article_ids)


    return article_ids, article_names # return the ids and names
```

[36]:
```
def user_user_recs(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user
```

```
    OUTPUT:
    recs - (list) a list of recommendations for the user

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides␣
↪them as recs
    Does this until m recommendations are found

    Notes:
    Users who are the same closeness are chosen arbitrarily as the 'next' user

    For the user where the number of recommended articles starts below m
    and ends exceeding m, the last items are chosen arbitrarily

    '''
    # Your code here
    # store recs
    recs = []

    # similar users with the given user_id
    most_similar_users = find_similar_users(user_id)

    # articles seen based on user_id
    articles_1 = np.array(get_user_articles(user_id)[0])
    # loop through each user
    for user in most_similar_users:
        # get articles for the each closest user with closeness in decreasing␣
↪order
        similar_articles = np.array(get_user_articles(user)[0])

        # find articles which user hasn't ever seen
        articles_0 = np.setdiff1d(similar_articles, articles_1,␣
↪assume_unique=True).tolist()
        print(articles_0)
        # store in recs
        recs += articles_0
        # If there are more than
        if len(recs) >= m:
            break

    recs = recs[:m]

    return recs # return your recommendations for this user_id
```

[37]:
```python
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

```
[]
['2.0', '12.0', '14.0', '16.0', '26.0', '28.0', '29.0', '33.0', '50.0', '74.0',
'76.0', '108.0', '120.0', '124.0', '131.0', '164.0', '193.0', '194.0', '210.0',
'213.0', '221.0', '223.0', '236.0', '237.0', '241.0', '252.0', '253.0', '283.0',
'295.0', '299.0', '302.0', '316.0', '336.0', '337.0', '339.0', '348.0', '359.0',
'362.0', '367.0', '409.0', '422.0', '444.0', '477.0', '482.0', '510.0', '517.0',
'524.0', '617.0', '634.0', '641.0', '656.0', '658.0', '665.0', '682.0', '693.0',
'720.0', '721.0', '729.0', '744.0', '761.0', '800.0', '812.0', '821.0', '825.0',
'833.0', '843.0', '887.0', '939.0', '943.0', '952.0', '957.0', '967.0', '969.0',
'973.0', '996.0', '1000.0', '1014.0', '1025.0', '1051.0', '1101.0', '1148.0',
'1159.0', '1160.0', '1162.0', '1163.0', '1164.0', '1165.0', '1166.0', '1171.0',
'1172.0', '1176.0', '1181.0', '1276.0', '1277.0', '1291.0', '1298.0', '1299.0',
'1304.0', '1314.0', '1330.0', '1332.0', '1336.0', '1338.0', '1343.0', '1351.0',
'1354.0', '1357.0', '1360.0', '1364.0', '1366.0', '1367.0', '1386.0', '1393.0',
'1395.0', '1396.0', '1423.0', '1428.0', '1432.0']
```

[37]:
```
['got zip code data? prep it for analytics. - ibm watson data lab - medium',
 'timeseries data analysis of iot events by using jupyter notebook',
 'graph-based machine learning',
 'using brunel in ipython/jupyter notebooks',
 'experience iot with coursera',
 'the 3 kinds of context: machine learning and the art of the frame',
 'deep forest: towards an alternative to deep neural networks',
 'this week in data science (april 18, 2017)',
 'higher-order logistic regression for large datasets',
 'using machine learning to predict parking difficulty']
```

[38]:
```python
# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0',␣
↪'1427.0'])) == set(['using deep learning to reconstruct high-resolution␣
↪audio', 'build a python app on the streaming analytics service', 'gosales␣
↪transactions for naive bayes model', 'healthcare python streaming␣
↪application demo', 'use r dataframes & ibm watson natural language␣
↪understanding', 'use xgboost, scikit-learn & ibm watson machine learning␣
↪apis']), "Oops! Your the get_article_names function doesn't work quite how␣
↪we expect."
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing␣
↪(2015): united states demographic measures','self-service data preparation␣
↪with ibm data refinery','use the cloudant-spark connector in python␣
↪notebook']), "Oops! Your the get_article_names function doesn't work quite␣
↪how we expect."
assert set(get_user_articles(20)[0]) == set(['1320.0', '232.0', '844.0'])
```

```
assert set(get_user_articles(20)[1]) == set(['housing (2015): united states␣
↪demographic measures', 'self-service data preparation with ibm data␣
↪refinery','use the cloudant-spark connector in python notebook'])
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.
↪0', '1422.0', '1427.0'])
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct␣
↪high-resolution audio', 'build a python app on the streaming analytics␣
↪service', 'gosales transactions for naive bayes model', 'healthcare python␣
↪streaming application demo', 'use r dataframes & ibm watson natural language␣
↪understanding', 'use xgboost, scikit-learn & ibm watson machine learning␣
↪apis'])
print("If this is all you see, you passed all of our tests!  Nice job!")
```

```
If this is all you see, you passed all of our tests!  Nice job!
```

4. Now we are going to improve the consistency of the **user_user_recs** function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the **top_articles** function you wrote earlier.

[60]:
```
def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
            1's when a user has interacted with an article, 0 otherwise


    OUTPUT:
    neighbors_df - (pandas dataframe) a dataframe with:
                    neighbor_id - is a neighbor user_id
                    similarity - measure of the similarity of each user to the␣
↪provided user_id
                    num_interactions - the number of articles viewed by the␣
↪user - if a u

    Other Details - sort the neighbors_df by the similarity and then by number␣
↪of interactions where
                    highest of each is higher in the dataframe

    '''
```

```python
    # Your code here
    similar_user_idx = find_similar_users(user_id)
    similarity_score = [dot_prod_user[user_id][i] for i in similar_user_idx]
    temp = df.groupby('user_id')['article_id'].count()
    num_interactions = [temp.loc[i] for i in similar_user_idx]
    neighbors_df = pd.DataFrame({'neighbor_id': similar_user_idx, 'similarity':␣
↪similarity_score, 'num_interactions':num_interactions})

    return neighbors_df.sort_values(['similarity','num_interactions'],ascending␣
↪= False) # Return the dataframe specified in the doc_string
```

[61]:

```python
get_top_sorted_users(1)
```

[61]:

	neighbor_id	similarity	num_interactions
0	3933	35	45
1	23	17	364
2	3782	17	363
3	203	15	160
4	4459	15	158
…	…	…	…
5138	2928	0	1
5139	2927	0	1
5140	2923	0	1
5144	2918	0	1
5147	2575	0	1

[5148 rows x 3 columns]

[62]:

```python
def user_user_recs_part2(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user by article id
    rec_names - (list) a list of recommendations for the user by article title

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides␣
↪them as recs
    Does this until m recommendations are found

    Notes:
    * Choose the users that have the most total article interactions
    before choosing those with fewer article interactions.
```

```
    * Choose articles with the articles with the most total interactions
    before choosing those with fewer total interactions.

    '''
    # store recs
    recs = []

    # similar users with the given user_id
    most_similar_users = get_top_sorted_users(user_id)['neighbor_id']

    # articles seen based on user_id
    articles_1 = np.array(get_user_articles(user_id)[0])
    # loop through each user
    for user in most_similar_users:
        # get articles for the each closest user with closeness in decreasing␣
↪order
        similar_articles = np.array(get_user_articles(user)[0])

        # find articles which user hasn't ever seen
        articles_0 = np.setdiff1d(similar_articles, articles_1,␣
↪assume_unique=True).tolist()
        articles_0 = df.groupby(['article_id'])['user_id'].
↪count()[list(map(float,articles_0))].sort_values(ascending=False).index.
↪tolist()

        # store in recs
        recs += articles_0
        # If there are more than
        if len(recs) >= m:
            break

    recs = recs[:m]
    # get rec_names
    rec_names = get_article_names(recs)


    return recs, rec_names
```

[63]:
```
user_user_recs_part2(10)
```

[63]:
```
([1431.0,
  1364.0,
  1293.0,
  1162.0,
  1436.0,
  1351.0,
```

```
  1393.0,
  1160.0,
  1354.0,
  1368.0],
 ['the nurse assignment problem',
  'predicting churn with the spss random tree algorithm',
  'analyze energy consumption in buildings',
  'visualize car data with brunel',
  'putting a human face on machine learning',
  'welcome to pixiedust',
  'model bike sharing data with spss',
  'finding optimal locations of new store using decision optimization',
  'analyze accident reports on amazon emr spark',
  'movie recommender system with spark machine learning'])
```

[64]:
```
# Quick spot check - don't change this code - just use it to test your functions
rec_ids, rec_names = user_user_recs_part2(20, 10)
print("The top 10 recommendations for user 20 are the following article ids:")
print(rec_ids)
print()
print("The top 10 recommendations for user 20 are the following article names:")
print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
[1330.0, 1427.0, 1364.0, 1170.0, 1162.0, 1304.0, 1351.0, 1160.0, 1354.0, 1368.0]

The top 10 recommendations for user 20 are the following article names:
['apache spark lab, part 1: basic concepts', 'predicting churn with the spss
random tree algorithm', 'analyze energy consumption in buildings', 'use xgboost,
scikit-learn & ibm watson machine learning apis', 'putting a human face on
machine learning', 'gosales transactions for logistic regression model',
'insights from new york car accident reports', 'model bike sharing data with
spss', 'analyze accident reports on amazon emr spark', 'movie recommender system
with spark machine learning']
```

5. Use your functions from above to correctly fill in the solutions to the dictionary below. Then test your dictionary against the solution. Provide the code you need to answer each following the comments below.

[65]:
```
get_top_sorted_users(1).loc[0,'neighbor_id']
```

[65]: 3933

[66]:
```
### Tests with a dictionary of results

user1_most_sim = get_top_sorted_users(1).loc[0,'neighbor_id']# Find the user␣
 ↪that is most similar to user 1
```

```
user131_10th_sim = get_top_sorted_users(131).loc[9,'neighbor_id']# Find the␣
↪10th most similar user to user 131
```

[67]:
```
## Dictionary Test Here
sol_5_dict = {
    'The user that is most similar to user 1.': user1_most_sim,
    'The user that is the 10th most similar to user 131': user131_10th_sim,
}

t.sol_5_test(sol_5_dict)
```

```
This all looks good!  Nice job!
```

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

Given that its a new user, i do not think user-user based collaborative filtering method, infact any collaborative filtering method would work over here, and both "user_user_recs" and "user_user_recs_part2" wont work in that case as the new user would not have any similar users to choose from. And in that case the only thing we can do is recommend the user top articles from every category. OR we can build a knowledge based system where the user specifies what he is interested in.

7. Using your existing functions, provide the top 10 recommended articles you would provide for the a new user below. You can test your function against our thoughts to make sure we are all on the same page with how we might make a recommendation.

[47]:
```
new_user = '0.0'

# What would your recommendations be for this new user '0.0'?  As a new user,␣
↪they have no observed articles.
# Provide a list of the top 10 article ids you would give to
new_user_recs = get_top_article_ids(10)# Your recommendations here
```

[48]:
```
assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.
↪0','1364.0','1304.0','1170.0','1431.0','1330.0']), "Oops!  It makes sense␣
↪that in this case we would want to recommend the most popular articles,␣
↪because we don't know anything about these users."

print("That's right!  Nice job!")
```

```
That's right!  Nice job!
```

1.1.8 Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the **doc_body**,

doc_description, or **doc_full_name**. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

1. Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

1.1.9 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
[ ]: def make_content_recs():
         '''
         INPUT:

         OUTPUT:

         '''
```

2. Now that you have put together your content-based recommendation system, use the cell below to write a summary explaining how your content based recommender works. Do you see any possible improvements that could be made to your function? Is there anything novel about your content based recommender?

1.1.10 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

Write an explanation of your content based recommendation system here.

3. Use your content-recommendation system to make recommendations for the below scenarios based on the comments. Again no tests are provided here, because there isn't one right answer that could be used to find these content based recommendations.

1.1.11 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
[ ]: # make recommendations for a brand new user


     # make a recommendations for a user who only has interacted with article id␣
      ↪'1427.0'
```

1.1.12 Part V: Matrix Factorization

In this part of the notebook, you will build use matrix factorization to make article recommendations to the users on the IBM Watson Studio platform.

1. You should have already created a **user_item** matrix above in **question 1** of **Part III** above. This first question here will just require that you run the cells to get things set up for the rest of **Part V** of the notebook.

[49]:
```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

[50]:
```
# quick look at the matrix
user_item_matrix.head()
```

[50]:
```
article_id  0.0  100.0  1000.0  1004.0  1006.0  1008.0  101.0  1014.0  1015.0  \
user_id
1           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
2           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
3           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
4           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
5           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0

article_id  1016.0  …  977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
user_id             …
1              0.0  …    0.0   0.0    1.0    0.0    0.0    0.0    0.0
2              0.0  …    0.0   0.0    0.0    0.0    0.0    0.0    0.0
3              0.0  …    1.0   0.0    0.0    0.0    0.0    0.0    0.0
4              0.0  …    0.0   0.0    0.0    0.0    0.0    0.0    0.0
5              0.0  …    0.0   0.0    0.0    0.0    0.0    0.0    0.0

article_id  993.0  996.0  997.0
user_id
1             0.0    0.0    0.0
2             0.0    0.0    0.0
3             0.0    0.0    0.0
4             0.0    0.0    0.0
5             0.0    0.0    0.0

[5 rows x 714 columns]
```

2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

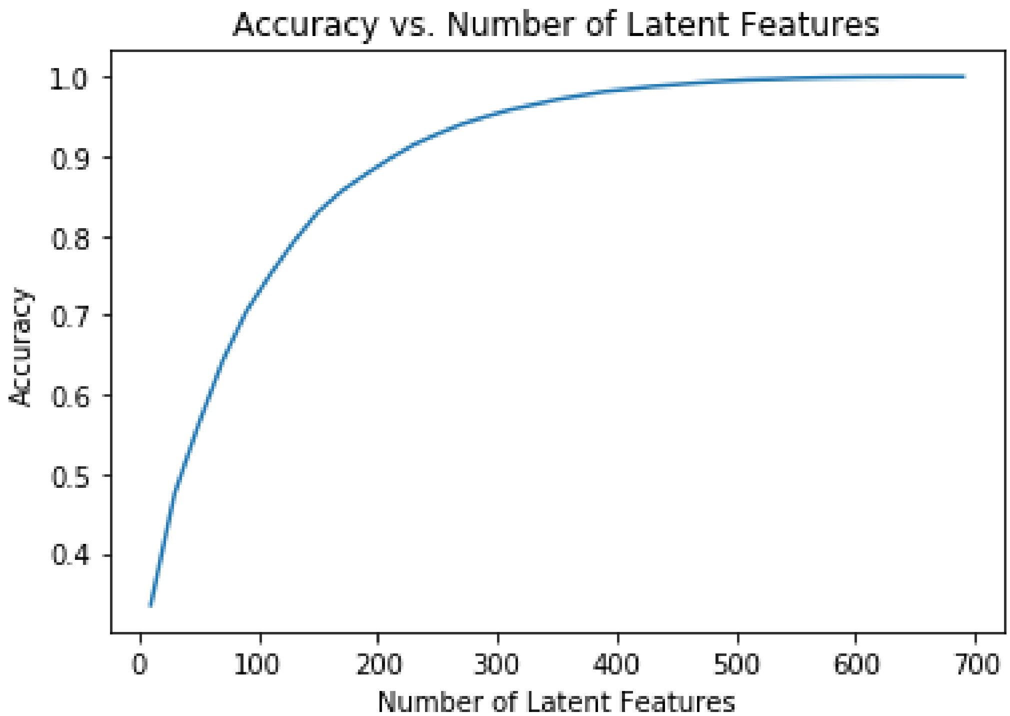
[51]:
```
# Perform SVD on the User-Item Matrix Here

u, s, vt = np.linalg.svd(user_item_matrix)
s.shape, u.shape, vt.shape# use the built in to get the three matrices
```

[51]: ((714,), (5149, 5149), (714, 714))

This is pretty obvious, in previous lesson we dumped SVD becuase of the only reason that it does not work with data where there are null values and we adopted FunkSVD, but in this case there are no null values so we might as well see how SVD works on this data.

3. Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

[52]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

    # take dot product
    user_item_est = np.around(np.dot(np.dot(u_new, s_new), vt_new))

    # compute error for each prediction to actual value
    diffs = np.subtract(user_item_matrix, user_item_est)

    # total errors and keep track of them
    err = np.sum(np.sum(np.abs(diffs)))
    sum_errs.append(err)


plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0]);
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
```



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

Use the code from question 3 to understand the impact on accuracy of the training and test sets of data with different numbers of latent features. Using the split below:

- How many users can we make predictions for in the test set?
- How many users are we not able to make predictions for because of the cold start problem?
- How many articles can we make predictions for in the test set?
- How many articles are we not able to make predictions for because of the cold start problem?

[69]:
```
df_train = df.head(40000)
df_test = df.tail(5993)
```

[72]:
```
df_test.groupby('user_id').count().shape
```

[72]: (682, 2)

[55]:
```
def create_test_and_train_user_item(df_train, df_test):
    '''
    INPUT:
```

```
    df_train - training dataframe
    df_test - test dataframe

    OUTPUT:
    user_item_train - a user-item matrix of the training dataframe
                      (unique users for each row and unique articles for each␣
↪column)
    user_item_test - a user-item matrix of the testing dataframe
                    (unique users for each row and unique articles for each␣
↪column)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''
    # Your code here
    user_item_train = create_user_item_matrix(df_train)
    user_item_test = create_user_item_matrix(df_test)
    test_idx = df_test['user_id'].unique()
    test_arts = df_test['article_id'].unique()

    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts =␣
↪create_test_and_train_user_item(df_train, df_test)
```

[56]:
```
#answer to How many users can we make predictions for in the test set?
a = len(set(user_item_train.index.values).intersection(set(user_item_test.index.
↪values)))
a
```

[56]: 20

[57]:
```
# answer to How many users in the test set are we not able to make predictions␣
↪for because of the cold start problem?
len(set(user_item_test.index.values)) - a
```

[57]: 662

[58]:
```
# answer to How many articles can we make predictions for in the test set?
user_item_train.columns.values.shape, user_item_test.columns.values.shape
```

[58]: ((714,), (574,))

[59]:
```
# Replace the values in the dictionary below
a = 662
b = 574
c = 20
```

```
d = 0


sol_4_dict = {
    'How many users can we make predictions for in the test set?':c,
    'How many users in the test set are we not able to make predictions for␣
 ↪because of the cold start problem?':a,
    'How many articles can we make predictions for in the test set?':b,
    'How many articles in the test set are we not able to make predictions for␣
 ↪because of the cold start problem?':d
}

t.sol_4_test(sol_4_dict)
```

```
Awesome job!  That's right!  All of the test movies are in the training data,
but there are only 20 test users that were also in the training set.  All of the
other users that are in the test set we have no data on.  Therefore, we cannot
make predictions for these users using SVD.
```

5. Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based on the accuracy on the test data. This will require combining what was done in questions 2 - 4.

Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

[60]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train, full_matrices=0)#␣
 ↪fit svd similar to above then use the cells below
```

[61]:
```
u_train.shape, s_train.shape, vt_train.shape
```

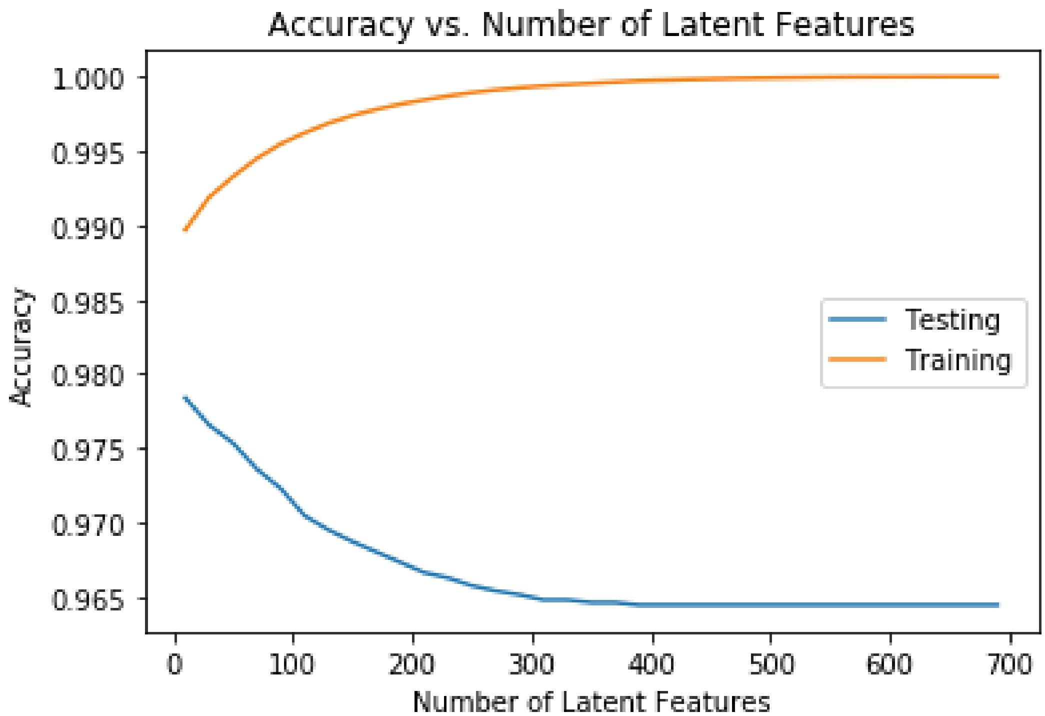
[61]:
```
((4487, 714), (714,), (714, 714))
```

[62]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data
```

[63]:
```
common_users = user_item_train.index.isin(test_idx)
u_test = u_train[common_users,:]
common_articles=user_item_train.columns.isin(test_arts)
vt_test = vt_train[:,common_articles]
# users on which we can make prediction from the test set.
final_test_users= user_item_test.index.isin(set(user_item_train.index).
 ↪intersection(user_item_test.index))
user_item_test_main = user_item_test[final_test_users]
```

[70]:

```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []
sum_errs_test = []


for k in num_latent_feats:
    # restructure with k latent features
    s_train_new, u_train_new, vt_train_new = np.diag(s_train[:k]), u_train[:, :
 ↪k], vt_train[:k, :]
    u_test_new, vt_test_new = u_test[:,:k], vt_test[:k,:]

    # take dot product
    user_item_est = np.around(np.dot(np.dot(u_train_new, s_train_new),␣
 ↪vt_train_new))
    user_item_est_test = np.around(np.dot(np.dot(u_test_new, s_train_new),␣
 ↪vt_test_new))

    # compute error for each prediction to actual value
    diffs = np.subtract(user_item_train, user_item_est)
    diffs_test = np.subtract(user_item_test_main, user_item_est_test)

    # total errors and keep track of them
    sum_errs.append(np.sum(np.sum(np.abs(diffs))))
    sum_errs_test.append(np.sum(np.sum(np.abs(diffs_test))))

plt.plot(num_latent_feats, 1 - np.array(sum_errs_test)/(user_item_test_main.
 ↪shape[0] * user_item_test.shape[1]), label = 'Testing');
plt.plot(num_latent_feats, 1 - np.array(sum_errs)/(user_item_train.shape[0] *␣
 ↪user_item_test.shape[1]), label = 'Training');
plt.xlabel('Number of Latent Features');
plt.legend()

plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
```



`6.` Use the cell below to comment on the results you found in the previous question. Given the circumstances of your results, discuss what you might do to determine if the recommendations you make with any of the above recommendation systems are an improvement to how users currently find articles?

I would like to take this question as an opportunity to summarize what we did in this project, and discuss some of the ways we can improve it:

- Load of functions were written to make the recommendation engine give wholistic reccomendation to the all kinds of users.
- If its a frequent user then we used the collaborative filtering methods, in this case it was user-user collaborative filtering method, where we found out the similarity of users based on the articles they interacted with. OR we can use SVD based method, here we used used normal SVD and not funkSVD as there were no null values in the dataset.
- But both of the above methods have their own downfall. For collaborative filtering approach we cannot tell how the model is doing without deploying it online. The SVD method covers up for that but despite of that both of these methods fail miserably for the users that have never been seen before.
- For the SVD method, the test set even though had 682 users, we could only make predictions for 20 of them as other 662 were not present in the test set.
- From the plot above we can observe that with increase in latent features the accuracy on the train set increases but the same pattern is not observed with the test set, although the decrease in accuracy is not that much to care about(0.980 to 0.965 not a big difference to care about).
- One of the possible reason for increase in accuracy in train set is the increase in the amount

of variability that can be captured by the increase in the number of latent features, but on the other hand the distribution of the test set is different from the train set(as we saw only 20 users were common in the train and test set). So even though the 20 users were also part of the train set the increase in number of latent features might be optimal for the whole dataset but might not be optimal for the subset of the data.

- And my opinion on the evaluation metric is somewhat biased towards Mean average precision(MAP) as it gives us an idea about the false positives and negative errors. Where as Accuracy is not capable of such analysis.
- We can make minor additions to the above methods, such as add a content based reccomendation system or just use the rank based knowledge system based on what the new user asks.
- But all of these additions are suboptimal as the recommendation system will still suffer from not having serendapity, novelty and diversity in it.
- To make up for that we can further hard code our system, that is if the user is watching too many movies of a certain category, we should put in a few recommendations apart from that category. For knowledge based systems we can calculate similarity between genres, example statistics is more close to machine learning and artificial intellence based articles, so instead of only recommending only statistics based articles to the user, we can slip in a few ML and AI based articles too. And same goes for software engineering articles too, based on the similarity Data structures based articles can be slipped in with them as the two genres are closer in similarity.

Extras Using your workbook, you could now save your recommendations for each user, develop a class to make new predictions and update your results, and make a flask app to deploy your results. These tasks are beyond what is required for this project. However, from what you learned in the lessons, you certainly capable of taking these tasks on to improve upon your work here!

1.2 Conclusion

Congratulations! You have reached the end of the Recommendations with IBM project!

Tip: Once you are satisfied with your work here, check over your report to make sure that it is satisfies all the areas of the rubric. You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

1.3 Directions to Submit

Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

Alternatively, you can download this report as .html via the **File > Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

[ ]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```